IMPLEMENTATION OF AN INTERACTIVE GRAPHICS DISPLAY IN A MULTIPROGRAMMING ENVIRONMENT

Lloyd Allen Thorpe

IMPLEMENTATION OF AN INTERACTIVE GRAPHICS DISPLAY IN A MULTIPROGRAMMING ENVIRONMENT

by

Lloyd Allen Thorpe

March 1976

Thesis Advisor: G. A. Raetz

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Implementation of an Interactive Graphics Display in a Multiprogramming Environment		5. TYPE OF REPORT & PERIOD COVERED Master's thesis; March 1976 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lloyd Allen Thorpe		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1976 13. NUMBER OF PAGES 27
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

graphics
interactive graphics
real-time processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis is a summary of the design and implementation of an operating system interface and a user interface for an interactive graphics display system. The actual interface software and documentation are characteristic of the Naval Postgraduate School environment. Documents describing the actual software and user interface are published separately.

(cont.)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont.)

The general problems and solutions involved in implementing a real-time interactive graphics process in a multiprogramming environment are included herein. The problems and solutions discussed are related to the interface of a Vector General Graphics Display Unit and a Digital Equipment Corporation PDP-11/50 computer. Recommendations for possible future developments are also included.

Implementation of an Interactive Graphics Display in a Multiprogramming Environment

by

Lloyd A. Thorpe
Lieutenant, United States Navy
B.S., University of Utah, 1971

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1976

ABSTRACT

This thesis is a summary of the design and implementation of an operating system interface and a user interface for an interactive graphics display system. The actual interface software and documentation are characteristic of the Naval Postgraduate School environment. Documents describing the actual software and user interface are published separately.

The general problems and solutions involved in implementing a real-time interactive graphics process in a multiprogramming environment are included herein. The problems and solutions discussed are related to the interface of a Vector General Graphics Display Unit and a Digital Equipment Corporation PDP-11/50 computer. Recommendations for possible future developments are also included.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I wish to thank my thesis advisor, Gary M. Raetz, for his effort and guidance. His knowledge and assistance has been a asset in the development of this thesis.

I wish to congratulate my wife, Linda, for enduring the long nights of study and the discouraging times of my frustration. Without her encouragement and helpfulness, the task would have been much more difficult. I also appreciate the tedious hours she spent in proofreading the initial manuscript and the helpful suggestions that resulted.

I. INTRODUCTION

The problem addressed in this thesis is how to support, in a multiprogramming environment, a process that does not conform to multiprogramming conventions. Four basic problems are identified in general and related to a specific interactive graphics environment in which they occur. The four problem areas are:

A. Operating System Modifications

B. Interrupt Interface Techniques

C. User Interface techniques

D. Accessing Non-Contiguous Display Lists

The vehicle used for problem identification and solution development was the Vector General Interactive Graphics Display System [5] and a Digital Equipment Corporation PDP-11/50 computer [17,18]. The UNIX Timesharing System [20] provided the multiprogramming environment.

The Vector General Interactive Graphics Display System (Vector General), as installed at the Naval Postgraduate School, is a highly sophisticated display terminal with hardware implemented three dimensional rotation, translation, and scaling [5]. An alphanumeric keyboard [1], lighted function switches with manual interrupt [4], control dials [2], and light pen [12] are attached to the system. A circle-arc generator [3] and a character generator [6] are included as an integral part of the system. This graphics display system is interfaced with a

PDP-11/50 computer having 64K bytes of memory and two million bytes of disk storage. This thesis discusses some of the problems involved in implementing an interactive graphics interface and includes recommendations for possible future developments in the area of supporting non-conforming processes in a multiprogramming environment. A detailed description of the actual interface designed and implemented in the course of this thesis can be found in separate publications [13,14]. These publications include a design manual, users manual, program listings, and documentation. The interface is a partial result and extension of an initial interface design by Howard and Thorpe [7,8].

II. CHARACTERISTICS OF MULTIPROGRAMMING PROCESSES

Multiprogramming is the interleaved or concurrent execution of two or more processes [13]. Since the physical memory requirements of each process may vary, each process is designed to be relocated within memory. This enables a process to be reassigned within memory as necessary to ensure efficient memory utilization. Processes waiting for a system resource are typically swapped onto a mass storage device thereby releasing the physical memory for another process. In some systems a process may be divided into segments or pages that are themselves relocatable and swappable entities.

A multiprogramming process typically is not given dedicated use of the central processor. Each process is executed for a time quantum and then set to a wait state. Processes waiting to be executed are placed in a queue according to some predefined priority. This is characteristic of computer systems permitting on-line communications with multiple users [11].

Since the multiprogramming process is relocatable and swapable, the user has no knowledge of physical memory address during program execution. Therefore, the user generates his program in an imaginary memory called virtual memory. Each users virtual memory begins at address zero and can, depending on the operating system, extend to the maximum address of the computer. The operating system maps all virtual addresses into physical addresses when the process is loaded into memory. Some of these characteristics conflict with characteristics of an interactive graphics process as will be shown.

III. CHARACTERISTICS OF INTERACTIVE GRAPHICS PROCESSES

A. CRT REFRESH REQUIREMENTS

While various types of graphics plotters have been invented, the CRT display is the only device suitable for generating interactive graphical output at high speed [16]. The short persistance of the CRT phosphor permits

information to be quickly changed. This is also the principle failing of the CRT. If a line is displayed once it quickly fades. The problem can be remedied by refreshing the CRT but continual refreshing limits the number of lines that can be drawn. If too many lines are displayed the intensity variations of the lines will be noticable. This phenomenon, called flicker, is undesirable and usually not permitted. A refresh rate of thirty to forty hertz will prevent flicker but does require the entire display to be displayed every thirty-three to twenty-five milliseconds. If the computer processor must be used to directly refresh the CRT display, supporting a graphics process under a multiprogramming environment would be impractical.

Recent developments in direct view storage tube displays and plasma displays offer a solution to the refresh problem but in some respects are less versatile than the conventional CRT. Another popular approach to the problem is to build a separate display processor whose function is to read the computer's memory and use the data to generate the display. Refresh processing is then handled by the display processor leaving the computer processor free to perform other tasks.

The display processor technique is employed by the Vector General using a Direct Memory Access channel (DMA) [5,6]. Communication with the user is maintained via a frame clock interrupt signal every 8.33 milliseconds.

Using the time between interrupts as a timer, the PDP-11/50 computer can determine when to initiate the Vector General for refresh. This allows the speed and versatility of a conventional CRT and removes the refresh processing from the PDP-11/50 processor. The DMA is not a panacea, however. It requires the display data to remain in the computer memory for the entire time the data is being used for display generation. While this is not a limitation on the computer processor, it is a limitation on the computer resources. This limitation would not be noticable in a dedicated computer environment. However, in a multiprogramming environment all computer resources are at a premium and and any limitations imposed must be considered.

B. MEMORY MANAGEMENT

As previously mentioned in section II, the central theme in a multiprogramming environment is that all active processed may be relocatable within memory. Processes may also be removed, or swapped onto a mass storage device. This permits efficient use of computer processor time and computer memory. But, this also conflicts with the DMA capability of the display processor.

Before the DMA can be used, some method of ensuring the entire display list is resident in memory must be found. In addition, the display list must not to be relocated or swapped. These restrictions, while necessary

to ensure the display processor can address the display list, must also be time minimized. Only while the display processor is actually using the display list for display generation must these restrictions apply. Therefore, some method of determining which display list is active must be found.

IV. GRAPHICS INTERFACE DESIGN

A. OPERATING SYSTEM MODIFICATIONS

Implementation of the graphics interface required that the memory allocation scheme of UNIX be modified to permit the Vector General Graphics Display System to access the graphics display list. The modification adopted involved creation of a unique real-time process. Whenever a user declares his intention to use the Vector General, the user's entire process is placed in physical memory and flagged as non-swapable and non-relocatable. Following the real-time process recommendations of Kral [10], the process priority is also increased to ensure the real-time process is at the head of the process queue. The process priority change has been found to be necessary only if the computer processor is performing the display refresh. It is not needed with DMA and its action prevents the computer processor from servicing any other process. The modifications implemented, while comparatively simple, do not consider memory as a limited resource nor is the time

minimization factor considered.

An alternate solution to the one implemented requires additional modifications to the operating system but does treat memory as a limited resource. This involves splitting a user process so that the instruction space and the data space are treated separately. When a user declares his intention to use the graphics terminal, the operating system places the restrictions only on the data space. The instruction space remains unchanged. This, however, still does not consider the time minimization factor. The user could declare his intentions to use the graphics terminal and then never do so. Yet, the resources would be allocated.

If a process were permitted to complete all calculations and data manipulations involving display list preparation prior to becoming a real-time process, the period of time in which memory is allocated to the real-time process would be reduced to the actual display time. The operating system would then, at display time, locate the display list referenced by the display request and make that list a real-time entity.

A third solution alternative is directly related to the capabilities of the Vector General Graphics Display System and the UNIX operating system. The address structure of the Vector General's DMA channel is such that the display processor can dynamically access only 32K bytes of memory. A display processor command defines

which 32K byte memory block is to be addressed.

The UNIX operating system treats the instruction and data space of a process as a single entity with the data space appended immediately following the instruction space. If the virtual addressing scheme of UNIX were modified so that the first address of the data space was virtual address zero, the real-time process modifications would be simplified. At display time, the process could be allocated across a 32K byte memory address boundary with its data space beginning at the boundary address. Virtual addresses would then be a physical address offset from the address boundary. While not providing a tremendous asset for memory management, this solution would greatly enhance the display manipluation capabilities.

A less appealing but easier implementation of the virtual addressing modification would be to ensure the entire virtual address space, instruction and data, begins on a 32K byte memory block boundary. The data space, while still an address offset, would not be virtual address zero. This implementation would reduce the total available data space by the size of the instruction space but the additional memory space used is that of the real-time process. Therefore, any addressing errors within the Vector General would only affect the instruction and data space of the real-time process.

B. INTERRUPT INTERFACE TECHNIQUES

A number of graphics devices have been invented for the input of graphical information to a computer. When used with a graphics display, the devices make it possible to effectively interact with the program. Generally, the simplest way to handle inputs from these devices is by means of interrupt routines which receive the input data and pass it on to user the program in the form of an interrupt signal [16]. Since the user exists in the relocatable process space of a virtual machine, the operating system must provide the link between the interrupt service routine and the user interrupt signal routine. This operation requires some effort and may take as much as one hundred milliseconds if the process has to be retreived from the disk. Considering the frequency of frame clock interrupts (8.33 milliseconds) plus the occurance of any device interrupts, it is likely that multiple interrupts will occur while waiting for the operating system to process the first interrupt.

The UNIX operating system has no capability of handling multiple interrupts nor of determining the priority of interrupts received from the same peripheral device. Because of the nature of some graphics devices, only one interrupt may occur. Therefore, it is important to ensure the preservation of each interrupt. The Vector General interface employs a technique that eliminates the

need to pass all but device interrupts to the user. The user is required to provide the interrupt service routine with the desired refresh rate. This determines the number of frame clock interrupts permitted before reinitialization of the display list. The interrupt service routine then handles the refresh timing without requiring any action from the user program.

When a device interrupt occurs, the values of specific Vector General registers are extracted prior to asking the operating system to signal the user program. These values representing the interrupt state of the Vector General are retained by the device interrupt service routine until the user program explicitely asks for the values. Included in these interrupt values is a Vector General status word indicating which interrupts have occured [5]. This feature enables the user to define which device has priority and any desired action to be taken.

C. INTERFACE TECHNIQUES

The user interface software has been designed to make the detailed operation of the Vector General transparent to the user. The basic concept is to define high level constructs which the user interface routines convert into Vector General commands. There are three classes of constructs defined: objects, elements, and the picture. An object is the lowest level construct which can be displayed alone. Each object is independently rotatable,

scalable, and translatable into any portion of a thirty inch by thirty inch picture space. An object can be as large as fifteen inches by fifteen inches and be rotated or positioned to the extreme limits of the picture space without distortion to any of the remaining visible portion. Each object is composed of one or more independently light pen hookable elements. An element is composed of a series of user drawn images or characters entirely relative to the untransformed image space of its object. An object can be defined unrotated in such a way as to fill the entire object space and then be scaled, rotated, and moved so that the image space is the appropriate size, is viewed from the appropriate aspect, and is in the appropriate area of the picture. The picture defines the picture scale and screen coordinates for all objects.

The user is responsible for the generation and content of each element. Prior to its inclusion within the display list, the user must fill each element with the necessary draw and move commands. In addition, the user must provide three unused words succeeding the draw-move commands. These three words are used by the interface routines to ensure each element is properly terminated. This prevents the Vector General from accessing memory outside the display list if the user fails to properly terminate the display list.

The generation and content of all objects and the

picture is the responsibility of the interface software. A set of routines are provided to link elements to objects and objects to the picture. Dynamic modification of objects and picture parameters is also provided. However, it is the user's responsibility to dynamically modify the element content.

D. ACCESSING NON-CONTIGUOUS DISPLAY LISTS

One of the basic requirements of an interactive graphics system is that the picture can be changed dynamically. This can be done by regenerating the entire display list or segmenting the display list and regenerating the modified segment. Segmenting the display list also permits sharing of display code in a manner somewhat analogous to conventional subroutines. This does require some means of generating the display from non-contiguous display lists. A list of the non-contiguous display lists could be created and sent to the display processor or the non-contiguous display lists could be linked within the display list itself. The Vector General is capable of both types of operation. The latter method was used for the interface implementation. This method requires less communication with the PDP-11/50 computer.

The concept of display subroutine calls implies the ability to store the present state before performing the subroutine jump. This usually is implemented by a Last-In-First-Out stack [9]. The DMA is normally a

bidirectional channel, therefore the display processor has access to the computer's memory for manipulation of a stack. With appropriate instructions and addresses encoded in the display list, the display processor can now access non-contiguous display lists. Reference 19 contains a description of the Vector General subroutine stack.

Permitting the display processor to write into memory creates a data protection problem in a shared memory environment. The display processor cannot be allowed to write indiscriminately throughout memory. The integrity of the operating system and other user processes must be maintained. A stack area provided by the user would enable the display processor to determine where to address memory for all write operations. However, the DMA bypasses the operating system's underflow and overflow protection mechanism. Therefore, an addressing error could cause modification of the operating system or another user process. Some method must be used to limit the access range of the display processor. Bounds registers in the display processor is a reasonable solution. These registers, set by the computer processor, would limit the area of addressability by the display processor. If the bounds registers applied to read as well as write operations, unbounded display lists could also be easily detected. The Vector General's 32K byte memory addressing limitation is analogous to a set of

fixed boundary resisters. Since the typical display list does not require the full 32K bytes of memory, the display processor does have the potential of addressing outside the real-time process. The Vector General interface routines minimize the problem in three ways. First, the user himself does not define the subroutine stack nor does he use the subroutine stack directly. This is handled by the interface routines. Second, the stack has a software underflow mechanism provided by the interface routines. An underflow traps to a Vector General halt instruction. Third, each display list is terminated in such a manner that the display list cannot be accessed beyond its defined length.

Realistically, there is no way to share the same 32K byte memory block with a Vector General process and ensure one hundred percent integrity. A user could include commands in his display list that cause an undetected stack overflow or a jump to an area outside of the process limits. The Vector General instructions defining these actions are not make available to the user but a display error could result in such an instruction.

VI. RECOMMENDATIONS

A. UNIX MODIFICATIONS

1. Process Priority

The current implementation of the interactive graphics interface requires each process requesting use of the Vector General to change its priority as part of the real-time system call. This is a necessary requirement only for those real-time processes performing the refresh. The increased priority ensures the real-time process is placed at the top of the process run queue. The direct memory access capability of the Vector General does not require the process to refresh the display directly. Therefore, the increased priority of the real-time process is not needed for the Vector General. If the UNIX operating system were modified to allow a process to be real-time without changing its priority, the affect of a real-time process on the multi-user environment would be reduced.

2. Memory Allocation

The memory allocation scheme of UNIX presently requires both the instruction space and the data space to be loaded contiguously in memory. The only requirement for the Vector General real-time process is that the active display list (data) be locked within a 32K byte memory block. If a process could be split so only its data space was real-time, the system memory resources

currently allocated for the real-time process instruction space would be available for other uses. The size of the Vector General data space could also be increased, thereby permitting more complex display lists.

B. USER INTERFACE MODIFICATIONS

1. Picture Rotation

As mentioned earlier, the only display construct capable of being rotated is an object. At times the user may desire to rotate the entire picture. This capability should be provided. Implementation of this feature must ensure the capability of rotating objects is not impared.

2. Display Enable

A limitation discovered by personnel using the interface is the inability to temporarily prevent the display of an element or an object without actually deleting it from the display list. The Vector General has this capability but it is not extended to the user.

3. Increment Timing

The proposed design interface of Howard and Thorpe [7,8] included a motion feature that allowed the user to automatically have an object move across the screen. The user defined motion vector, using the frame clock interrupts as a timer, automatically incremented the position of specified objects by the values of the motion vector. The actual implementation could not include this feature because of the relationship between the user

process and the frame clock interrupts. The only timer UNIX has provided is in increments of one second and that timer is stopped by any interrupt signal. Therefore, it is entirely unsuited for a display motion timer. Implementation of this capability would enhance the use of the Vector General and simplify the user program.

4. Display List Generation

The importance of programming languages is often forgotten when a graphics system is designed. The designer becomes involved in the issues of display file structures and graphical interaction leaving the provision of a convenient programming language until later. This lack of interest in the development of programming languages has been one of the major obstacles preventing the widespread use of graphics [16]. Until such a language is developed, the unfortunate programmer is forced to write in machine or assembly language. This interface design has considered only part of the problem. Several interface routines have been provided to simplify the actual access to the Vector General. The creation of the display list used to generate pictures must still be accomplished in the octal machine language described by Thorpe [14,15]. Before the existing interface will be of use to the general graphics programmer, a simple method of generating display lists must be found. On this single capability may hinge the success of the existing interface structure.

VII. CONCLUSION

The Vector General interface designed and implemented by Thorpe [14,15] is operational with no known bugs in any of the interface routines. The hardware support and direct memory access capabilities of the Vector General Interactive Graphics Display System have been the key to a successful interface. Without the hardware support it is questionable whether any multi-user capabilities would be available while using the Vector General.

BIBLIOGRAPHY

1. Alphanumeric Keyboard KB1 Option Reference Manual, Vector General Inc., Woodland Hills, California May 1974

2. Analog Devices Option Reference Manual, Vector General Inc., Woodland Hills, California, August 1974

3. Circle-Arc Generator Reference Manual, Vector General Inc., Canoga Park, California, February 1974

4. Function Switch Option Reference Manual, Vector General Inc., Canoga Park, California, April 1974

5. Graphics Display System Reference Manual, Vector General Inc., Woodland Hill, California, August 1974

6. Graphics Display System Technical Manual, Vector General Inc., Woodland Hills, California, June 1974

7. Howard, J. E. and Thorpe, L. A., Proposed Design specification Manual for the Vector General Graphics Display Unit, paper presented at Naval Postgraduate School CS4204 Class, June 1975

8. Howard, J. E. and Thorpe, L. A., Proposed Users Manual for the Vector General Graphics Display Unit, paper presented at the Naval Postgraduate School CS4202 Class, June 1975

9. Knuth, D. E., The Art of Computer Programming, 2d ed., v. 1, Addison-Wesley, 1968

10. Kral, T. C., A Process Controller for a Heirarchical Process Structured Operating System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1975

11. Lancaster, F. W. and Fayen, E. G., Information Retrieval On-Line, Melville Publishing Company, 1973

12. Light Pen LP3 Option Reference Manual, Canoga Park, California, May 1974

13. Madnik, S. E. and Donovan, J. J., Operating Systems, McGraw-Hill Book Company, 1974

14. Naval Postgraduate School Technical Report NPS72Rr760302, Design Manual for the Interactive Graphics Display Unit, by L. A. Thorpe and G. M. Raetz, March 1976

15. Naval Postgraduate School Technical Report NPS72Rr76031, Users Manual for the Vector General Graphics Display Unit, by L. A. Thorpe and G. M. Raetz, March 1976

16. Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill Book Company, 1968

17. Peripherals Handbook, PDP-11, Digital Equipment Corporation, 1974.

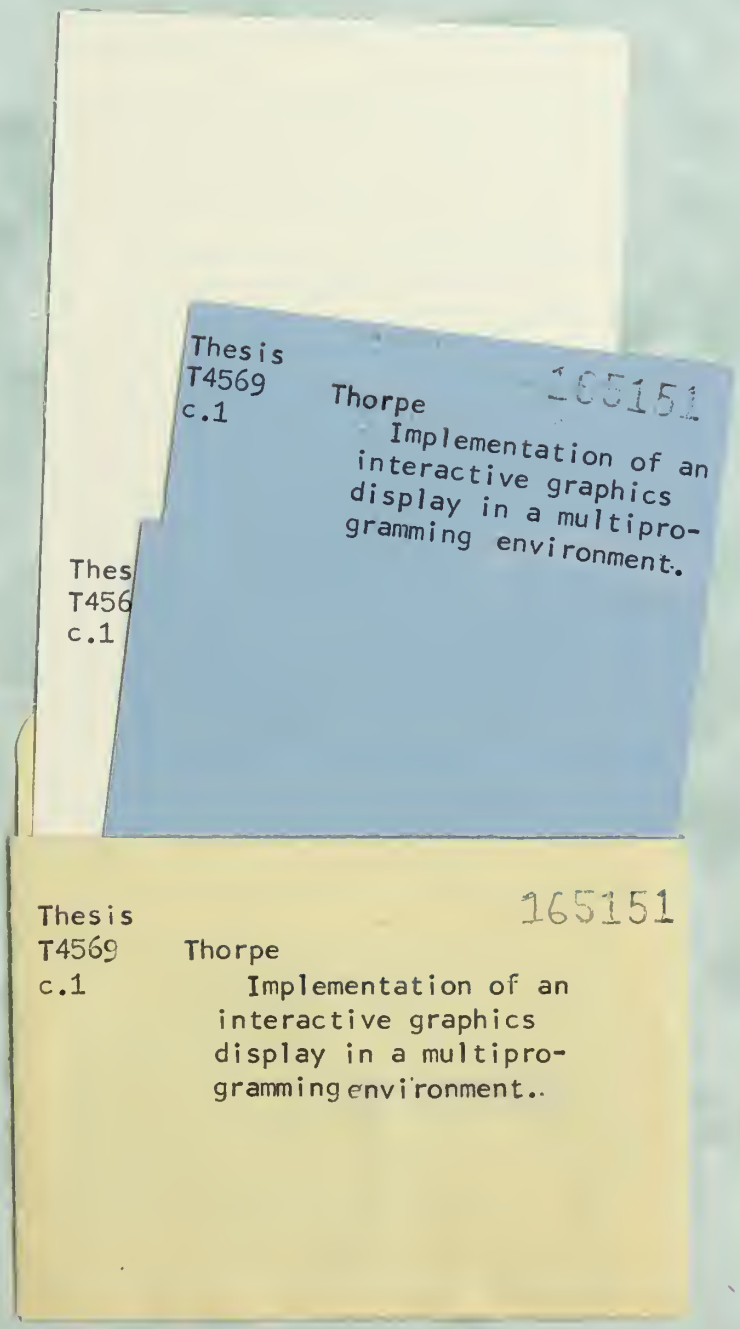
18. Processor Handbook, PDP-11/45, Digital Equipment Corporation, 1974.

19. PDP-11 Interface Option (with Sub-Stack) Reference Manual, Woodland Hills, California, August 1974

20. Ritchie, D. E. and Thompson, K., The UNIX Timesharing System, Communications of the ACM, v. 17, no. 7, p 365-375, July, 1974

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexanderia, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Professor George A. Rahe, code 72Ra Computer Science Group Naval Postgraduate School Monterey, California 93940	1
5.	Ltjg. Gary M. Raetz, USN, Code 72Rr Computer Science Group Naval Postgraduate School Monterey, California 93940	1
6.	Commanding Officer Naval Electronic Systems Command Code 320 Washington, D. C. 20360 Attn: CDR Miller	1
7.	Lt. Lloyd A. Thorpe, USN 915 24th Street West Billings, Montana 59102	1


Thesis
T4569
c.1
Thorpe
Implementation of an interactive graphics display in a multiprogramming environment.
165151
Thes
T456
c.1
Thesis
T4569
c.1
Thorpe
Implementation of an interactive graphics display in a multiprogramming environment..
165151